A write-performance improvement of ZABBIX with NoSQL databases and HistoryGluon

MIRACLE LINUX CORPORATION*

February 13, 2013

Abstract

This paper presents our research progress for improving write performance of ZABBIX with a NoSQL database for storing histories whose write frequency is increased as the number of monitoring targets. The benchmark results shows that write performance with our proposed method is increased by approximately three times compared to that with the original.

1 Introduction

Today the number of devices such as servers and network equipment in enterprise data center has been increasing due to advent of new technologies (e.g., cloud computing, PaaS and IaaS, and a framework for running applications on large cluster such as Hadoop [1]). Monitoring a huge number of devices is becoming more and more important.

ZABBIX [2] is one of the most popular open source monitoring software and can also be applied for an enterprise system [3]. As the number of monitoring targets increases, write frequency of histories that are monitored data such as CPU load, available memory size, free disk space, and so on becomes high. Therefore it is considered that write performance of histories is one of the significant factors to decide the maximum number of monitoring targets.

ZABBIX uses one of MySQL [4], PostgreSQL [5], Oracle [6], DB2 [7], and SQLite [8] as a DB (Data Base) to store both histories and other information such as monitoring items, intervals, trigger conditions, user names and so on. The all DBs described above are categorized into RDBMS (Relational Data Base Management System). It generally has an SQL interface and the functions required for SQL statements and has been widely used for various applications.

Recently a lot of open source products of NoSQL (Not Only SQL) that is a different type of DB have appeared. Some of them have faster read and/or write performance than RDBMS. We created a software stack that stores histories on a NoSQL DB instead of an RDBMS with the goal of write-performance improvement. This paper presents the architecture and the benchmark results.

2 Proposed architecture

Figure 1 shows a proposed software stack. There are two newly added components that are shown in gray boxes. One is a NoSQL DB and the other is HistoryGluon [9]. The fixed version of ZABBIX 2.0.3 [10] is used in the proposed software stack.

zabbix_server (DBSyncer) stores histories into a NoSQL DB via HistoryGluon that provides high level APIs to save history. The source code that originally uses SQL statements in zabbix_server is replaced by the source code that calls APIs of HistoryGluon client library. Front-end PHP source code also gets histories via HistoryGluon PHP extension.

2.1 NoSQL databases

A NoSQL DB is a DB that doesn't have an SQL Interface. There are many products of it in the open-source world. Some of them have simple data structure and works faster than RDBMS. In addition, they have a scale out feature that increases read and/or write performance by adding nodes (computers) to run themself. In this paper, the benchmark results with NoSQL DBs: HBase[11], Cassandra[12], and Riak[13] are reported. Their features are briefly summarized in Table 1.

*The lead author: Kazuhiro Yamato, e-mail: kazuhiro.yamato@miraclelinux.com

[1]Single Point Of Failure

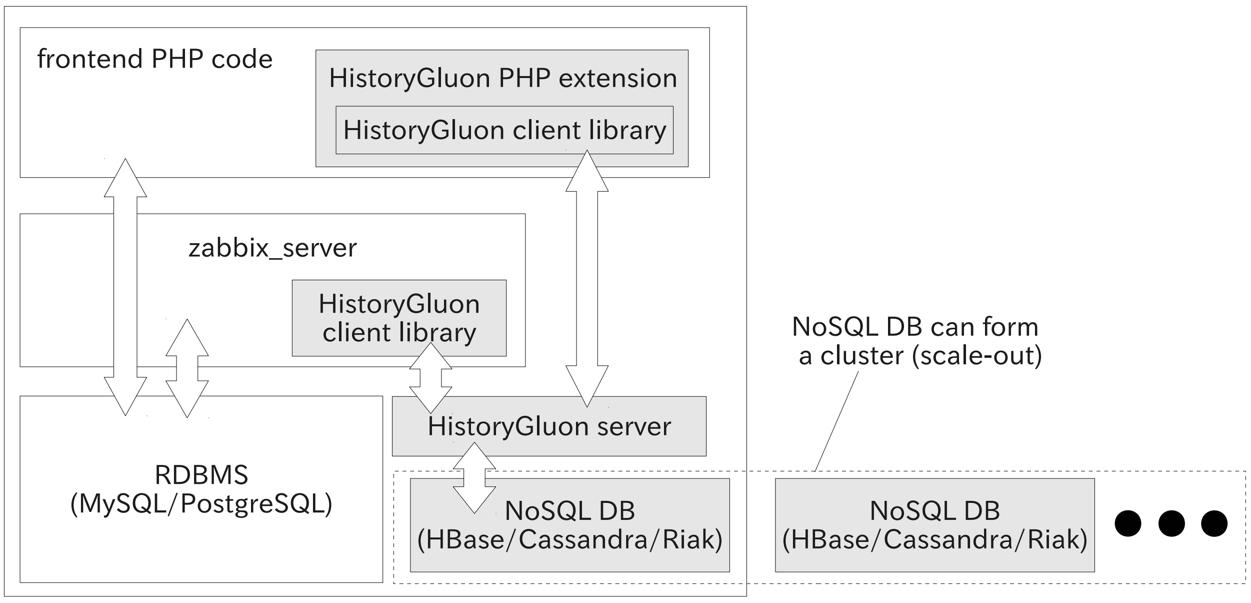
frontend PHP code
HistoryGluon PHP extension
HistoryGluon client library
zabbix_server
HistoryGluon client library
NoSQL DB can form a cluster (scale-out)
HistoryGluon server
RDBMS (MySQL/PostgreSQL)
NoSQL DB (HBase/Cassandra/Riak)
NoSQL DB (HBase/Cassandra/Riak)

Figure 1: A proposed software stack. A gray box indicates a newly added component.

Table 1: Comparison table of NoSQL DBs used in the benchmark.

	HBase	Cassandra	Riak
Developed by	Apache Software Foundation	Apache Software Foundation	Basho Technologies, Inc.
License	Apache License v2	Apache License v2	Apache License v2
Consistency	Strict	Eventual	Eventual
Have SPOF[1]	Yes	No	No
Version	0.94.1	1.1.6	1.2.1

2.2 HistoryGluon

HistoryGluon provides functions for storing/getting histories into/from various NoSQL DBs with a simple way from some programming languages. It has been on public at [9] under the license GPL version 3. We originally developed it to make it easy to evaluate the combination of ZABBIX and some NoSQL DBs. If there isn't a component that plays the role, it is needed to fix zabbix_server with respect to each NoSQL DB because NoSQL DBs don't have a common interface or an API set as its name suggests. However, now HistoryGluon is not only for ZABBIX, but also can be used for other applications because the API set has been designed for storing general histories.

HistoryGluon consists of a server and client libraries as shown in Table 2. The HistoryGluon server is written in Java. The primary reason is because many NoSQL DBs have high affinity with Java. In addition, it is one of the reasons that execution speed of a program written in Java is fast.

Figure 2 shows a class diagram related to the storage of histories in the HistoryGluon server. This structure enables to support multiple NoSQL DBs as a pluggable back-end DB. An interface `StorageDriver` defines abstract methods. Some of them are implemented in `BasicStorageDriver`. Classes that inherits it have implementation depending on a background NoSQL DB. They has been shortly written with the aid of common methods in `BasicStorageDriver`.

`HBaseDriver`, `CassandraDriver`, and `RiakDriver` are the driver to used HBase, Cassandra, and Riak as

Table 2: Components of HistoryGluon.

Component	Written in	Available from
server	Java	N/A (stand-alone program)
client library	C	C and C++
PHP extension	C	PHP

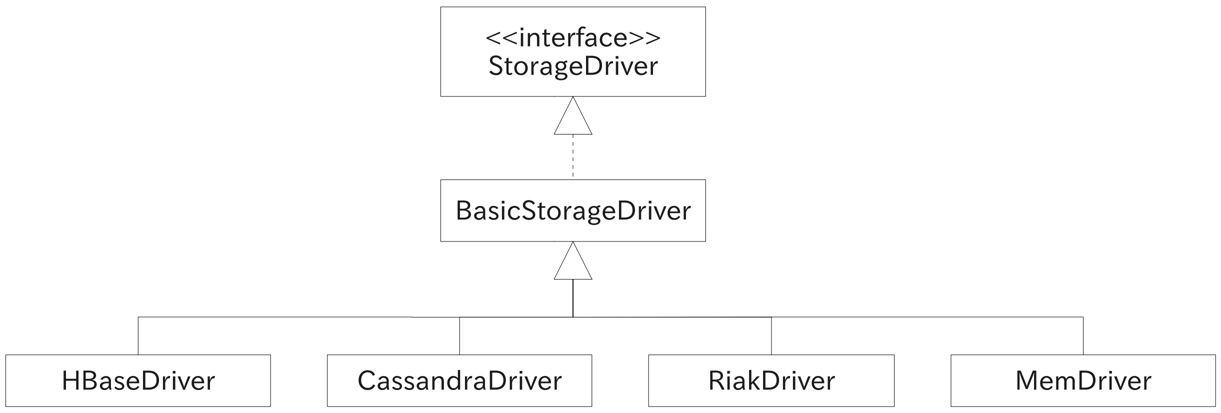

Figure 2: A class diagram related to the storage of histories.

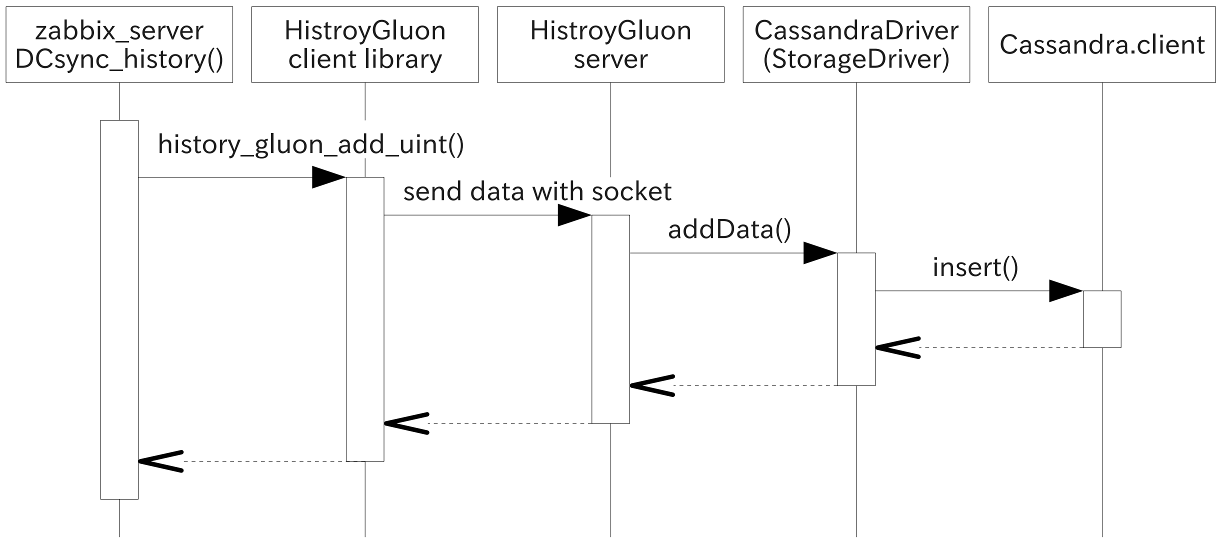

Figure 3: A sequence diagram in case a history with an integer value is stored on Cassandra.

a back-end DB respectively. `CassandraDriver` inserts histories with ConsistencyLevel.ONE. `MemDriver` stores histories only on memory with the Java's TreeSet class. It does not require any setting and can be used easily. Time to store and get a history is naturally short. It was developed mainly for a test.

Figure 3 shows a sequence diagram in case a history with an integer value is stored on Cassandra. As you can see, the current implementation calls functions synchronously. Since `addData()` is an abstract method defined in `StorageDriver` interface, left-hand components (i.e. zabbix_server, HistoryGluon client library and HistoryGluon server) are not aware which NoSQL DB is being used. The used NoSQL DB is specified in a command line to run HistoryGluon server.

HistoryGluon server can open and keep connections from multiple clients. Because a number of zabbix_server (DBSyncer) typically work in the system, the sequence shown in the figure is possible to be executed concurrently.

2.3 Other components for the benchmark

We added a function to zabbix_server to discard histories in DCsync_history(). When this function is enabled, a benchmark result corresponds to the performance in the extreme case that the time to write a history is zero. The result is informative as a reference.

3 Benchmark

3.1 Basic scheme

We use the number of histories written in DBSyncer as an index of the performance. DBSyncer originally logs the message including such information with the level LOG_LEVEL_DEBUG. We changed the level to LOG_LEVEL_INFORMATION to record it on the log file and added output points of the message (for more details, see the patch in [14]). Basic benchmark scheme is the following. It is repeated with increasing the number of monitoring targets. Actually they are automatically performed by the test script [14].

1. Register monitoring targets to zabbix_server with ZABBIX APIs.
2. Monitor the target items for T_{obs} sec.
3. Parse the log file and sum the written counts.
4. Remove the registered targets from zabbix_server.

In this benchmark, a lot of histories are generated by registering one target computer on which zabbix_agentd is running many times. A set of monitoring items are based on Template OS Linux. Monitoring interval is set to T_{int} sec. As long as the histories are normally saved, the number of written histories N should be approximately

$$N \simeq \frac{T_{obs}}{T_{int}}. \tag{1}$$

We performed the benchmark with $T_{obs} = 120$ and 1800 sec. and $T_{int} = 5$ sec.

As described in Section 1, ZABBIX supports many RDBMS. We use MySQL and PostgreSQL as ordinary RDBMS, because they are popular open source products and have standard client-server architecture.

3.2 Setup

Figure 4 and 5 show the setup for stand-alone mode and cluster mode respectively. In stand-alone mode, all components except for zabbix_agend are running on one node (node S). In cluster mode, programs of NoSQL DBs are distributed and executed. In both modes, zabbix_server, RDBMS (MySQL or PostgreSQL), and HistoryGluon server are executed on one node (node S). All nodes have the same specification listed in Table 3.

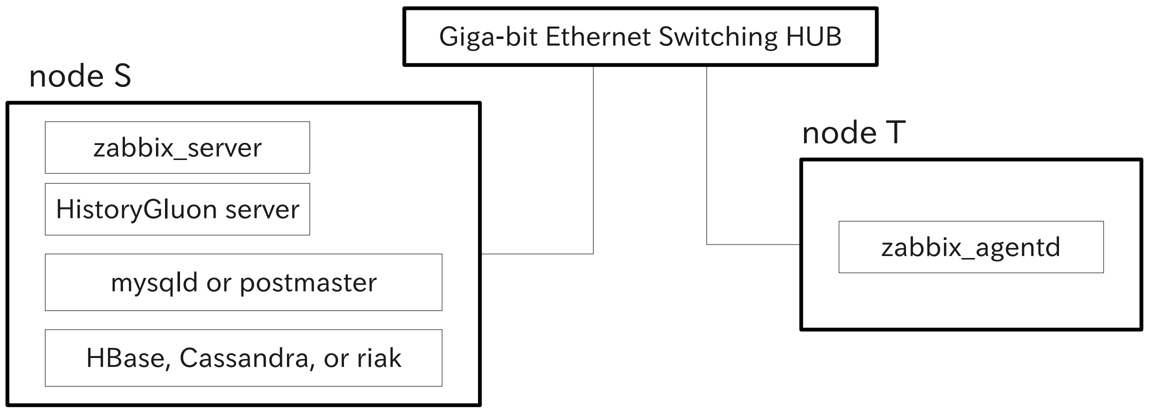

Figure 4: A setup for stand-alone mode.

3.3 Configurations of NoSQL DB

Default partitioner of Cassandra is RandomPartitioner. It evenly stores data to each node with a hash of the key. However this disables the range query with a key. Because ZABBIX often uses the range query, we set the partitioner to ByteOrderedPartitioner. With the partitioner, each node of Cassandra cluster has a range in which it stores data with a key. The lower boundary is called token. Cassandra provides the way to set the token of nodes by users. However, this partitioner may cause bias of node in which data is written. In this benchmark, we prepare three types of tokens. The first gathers all data to the local node (node S) on which

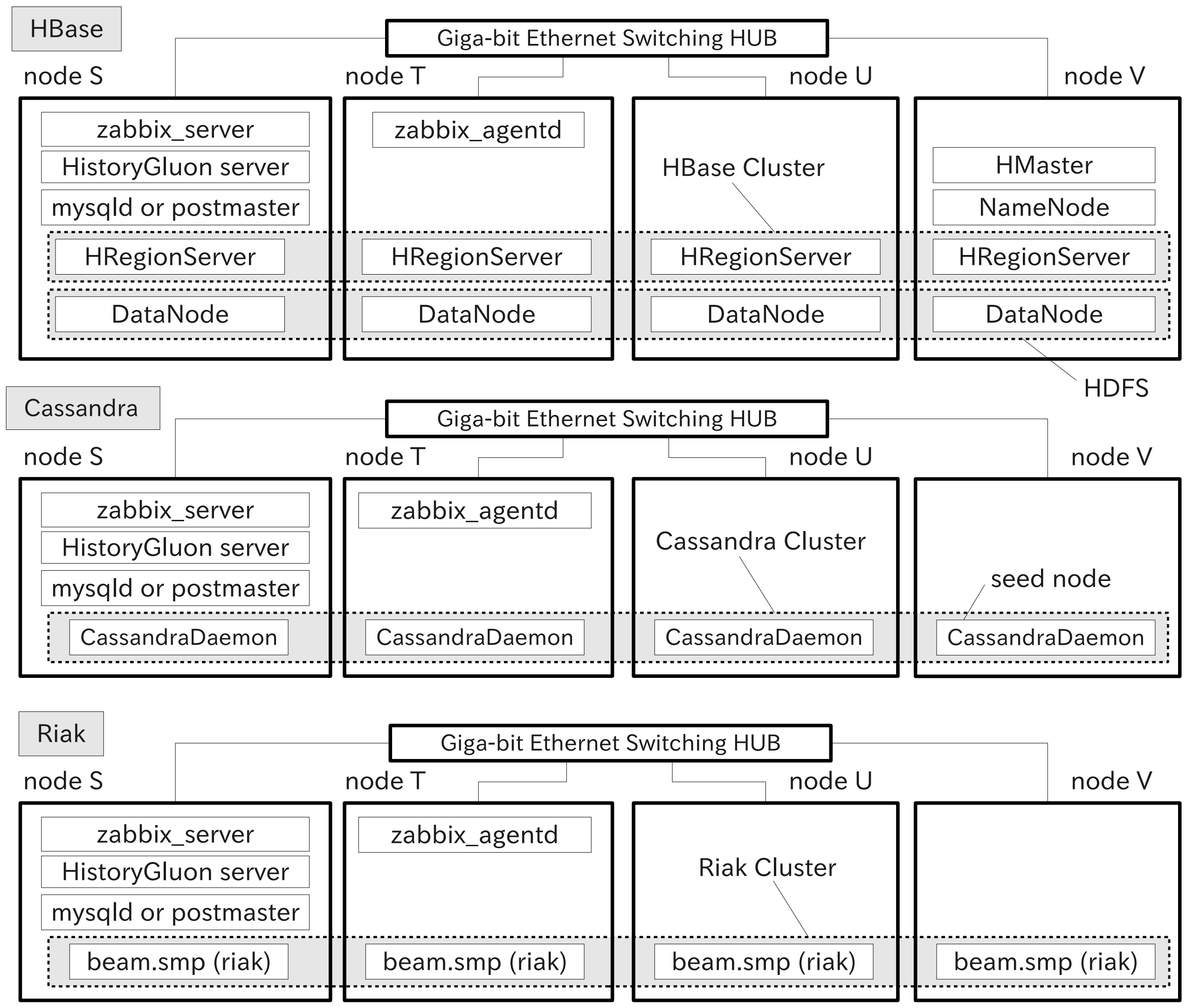

Figure 5: A setup for cluster mode.

HistoryGluon is running. The second gathers all data to an extern node (node U). The last evenly distributes data on each node.

Riak also cannot do the range query with the default storage back-end: riak_kv_bitcask_backend. We change the storage back-end to riak_kv_eleveldb_backend.

3.4 Configuration files

Configuration files for the benchmark are put in 'benchmark/conf' directory in [14].

4 Results and Discussion

Figure 6, 7, and 8 show the benchmark results. The horizontal and vertical axis indicate the number of monitoring targets and written history in T_{obs}, respectively. A slanted line passing through the origin means the ideal value when all histories are written without problems.

4.1 Stand-alone mode

In Figures 6 and 7, blue circles, magenta argyles, blue closed boxes, and black closed triangles are measured points with Cassandra, Riak, HBase, and Mem, respectively. The DB that stores information other than histories is MySQL or PostgreSQL respectively as shown in the figures. Red triangles and red upside-down triangles are measured points in the case that histories are also stored in RDBMS (MySQL and PostgreSQL

Table 3: Specification of PC and software used in the benchmark.

CPU	Intel Core i5-3570K 3.40GHz
Chipset	Intel Z77 Express
Memory	DDR3 1600MHz 4GB $\times 4$
HDD	Seagate ST2000DM001 formatted with EXT4 (No LVM is used)
LAN	Realtek RTL8111/8168B PCI Express Gigabit Ethernet controller
Distro.	CentOS 6.3 (64bit)
Java	OpenJDK 1.7

respectively). Black crosses are measured points in the case of Null Write mode. Hereafter we simply call the measured points Cassandra, Riak, HBase, Mem, MySQL, PostgreSQL, and Null Write as with the legend in the figures.

Mem, and Null Write are almost the same and on the ideal line up to where the number of targets is approximately 22000. The result of Cassandra with MySQL is also the same as those of Mem and Null. However, the result of Cassandra with PostgresSQL is lower than them.

Riak, MySQL, and PostgreSQL lost touch with the ideal line at the smaller number of monitoring targets (approximately 6000). That of HBase is remarkably the smallest in the benchmark. There aren't measured points of HBase above number of monitoring targets > 3000. In such conditions, we failed to measure the performance, because HistoryGluon crashed due to OutOfMemoryError of Java VM.

In all results, the number of written histories are almost constant value in the above a certain number of monitoring targets. We call the value a write-performance hereafter. A write-performance of Cassandra is more than 3 times of that of MySQL and PostgreSQL. This result implies that the bottleneck of a write-performance in the original ZABBIX is RDBMS.

In figure 6, the write-performance of Cassandra is very closely to that of Null Write and Mem. This suggests that the suppression factor in these condition is not performance of Cassandra. Currently the reason has not been revealed. It may be due to the setup method or an internal structure of zabbix_server and zabbix_agent.

4.2 Cluster mode

In Figure 8, black circles, blue upside-down triangles, and black triangles represent the measured points with Cassandra cluster. Hereafter we call them Cassandra (local), Cassandra (sharding), and Cassandra (external), respectively. They have the different tokens as described in section 3.3.

Cassandra (local) is comparable to the result of Cassandra in stand-alone mode. This is reasonable, because their data flow are almost the same. A write performance of Cassandra (external) is less than half of that of Cassandra (local). This may be caused by the simple synchronous architecture shown in Figure 3, which waits for the reply from an external node. A write performance of Cassandra (sharding) is located between that of Cassandra (local) and Cassandra (external). Magenta argyles and blue closed boxes represents the measured points with Riak cluster and HBase cluster respectively. The write-perfomance of Riak was improved by approximately 1.5 times compared to the stand-alone (Figure 6). However, other results of cluster mode have not a big difference from those of stand-alone mode with the current architecture and the setting.

5 Future plan

Here we enumerate items to propel this research below. We will keep trying the enhancement of performance by studying them.

- Understanding the cause of bottleneck in the result with Cassandra in stand-alone mode. If it is solved, the write-performance could be increased moreover.
- Benchmark of the read-performance. We will evaluate not only throughput but also the response time.
- Addition of supported NoSQL DBs such as MongoDB and the benchmark them.
- Addition of asynchronous HistoryGluon APIs and utilization of batch APIs of NoSQL DBs. These could improve the write performance in cluster mode.
- Understanding why the result with HBase was too low.

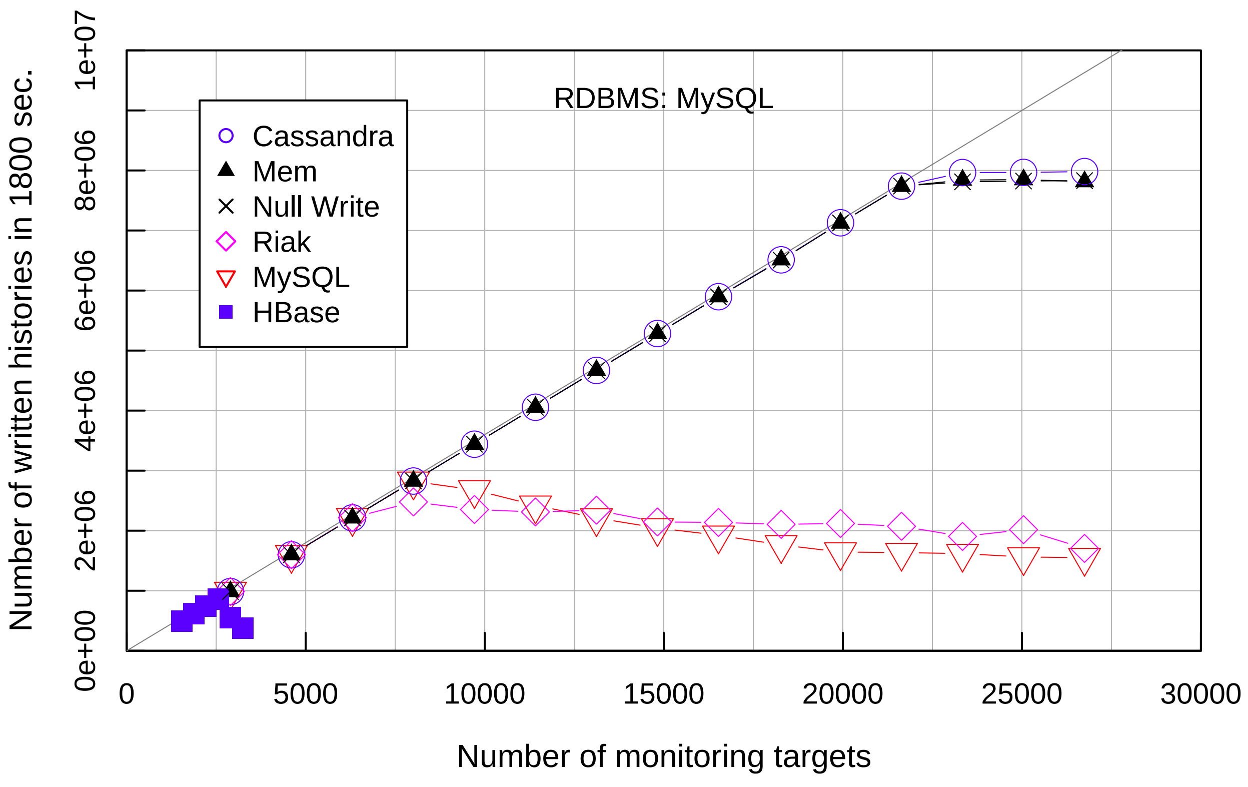

Figure 6: A summary of write performance (stand-alone, 1800sec.).

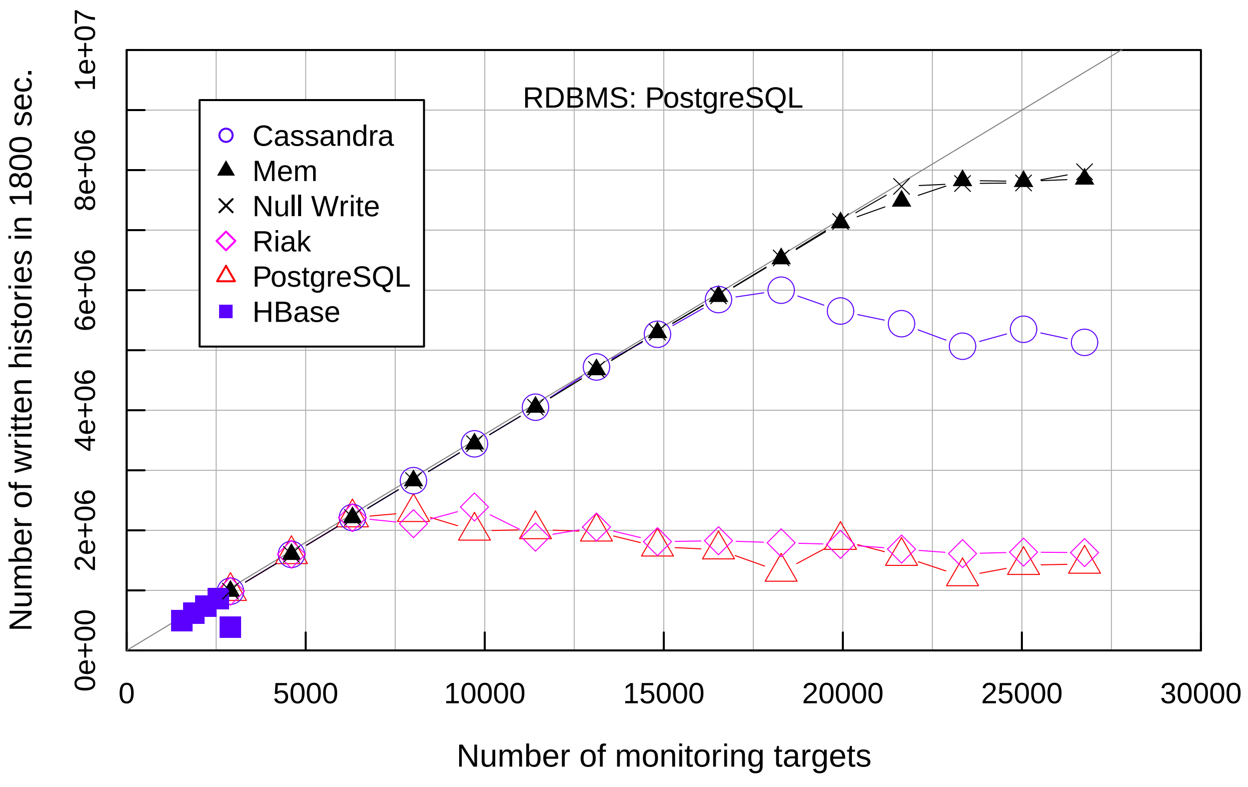

Figure 7: A summary of write performance (stand-alone, 1800sec.).

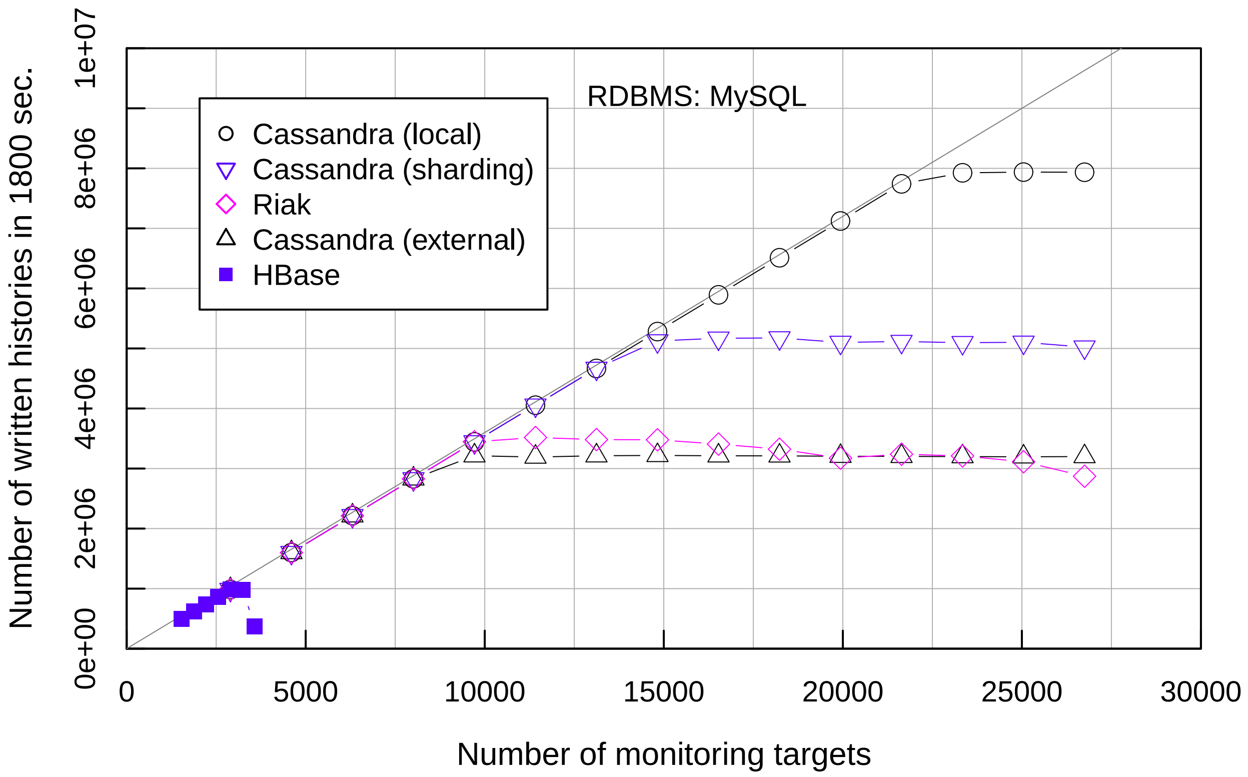

Figure 8: A summary of write performance (cluster, 1800sec.).

6 Summary

Monitoring software that can handle a huge number of devices is needed for these days. We proposed the method for improving a write-performance of ZABBIX by storing histories that was increased as the number of devices with a NoSQL DB and HistoryGluon, which was newly developed to become a bridge between NoSQL DBs and programs written in C such as zabbix_server. We evaluated the performance of the proposed method with NoSQL DBs: HBase, Cassandra, and Riak. The result with Cassandra has been increased approximately three times compared to original ZABBIX in the stand-alone mode. The result of the cluster with Riak has been increased approximately 1.5 times compared to that of stand-alone mode.

7 Contact and feedback information

Thank you for reading. You can rate this paper and contact us at [15].

References

[1] http://hadoop.apache.org/

[2] http://www.zabbix.com

[3] http://www.zabbix.com/enterprise_ready.php

[4] http://www.mysql.com/

[5] http://www.postgresql.org/

[6] http://www.oracle.com

[7] http://www-01.ibm.com/software/data/db2/linux-unix-windows/

[8] http://www.sqlite.org/

[9] https://github.com/miraclelinux/HistoryGluon

[10] https://github.com/miraclelinux/MIRACLE-ZBX-2.0.3-NoSQL

[11] http://hbase.apache.org

[12] http://cassandra.apache.org

[13] http://wiki.basho.com

[14] https://github.com/miraclelinux/zabbix-benchmark

[15] https://docs.google.com/forms/d/1BKroVJkb108CKHKBFfGaV25iE5zfF_R0bD0YSjyuhSw/viewform